Chancel: efficient multi-client isolation under adversarial programs

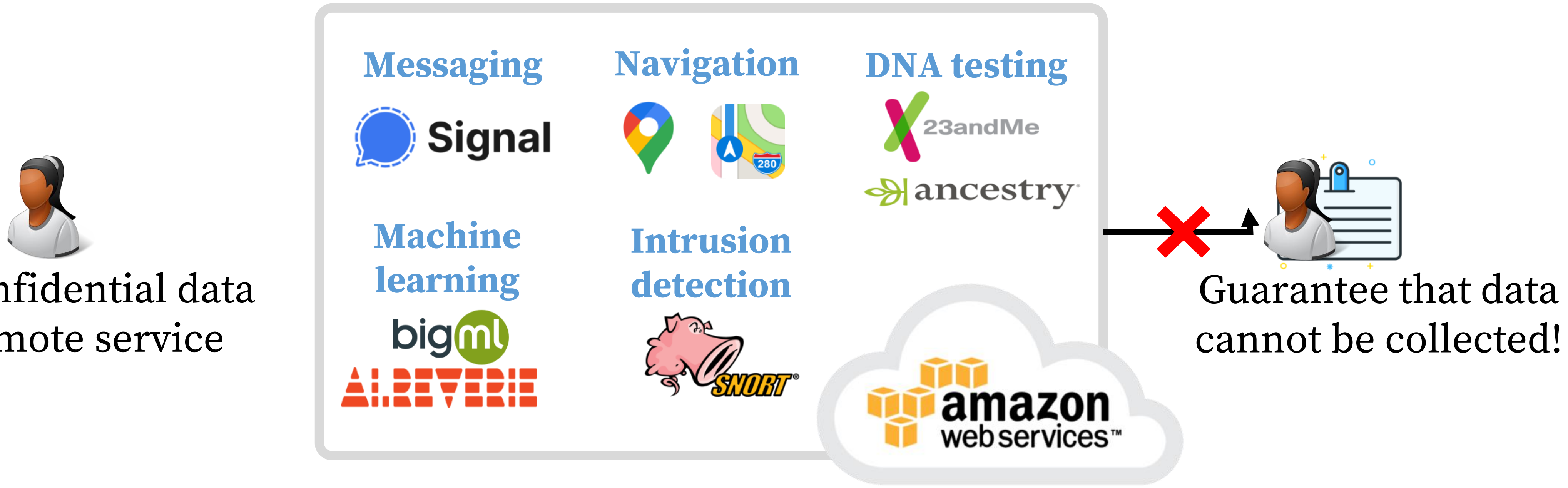
Adil Ahmad, Juhee Kim, Jaebaek Seo,
Insik Shin, Pedro Fonseca, and Byoungyoung Lee

PURDUE UNIVERSITY

SEOUL NATIONAL UNIVERSITY

KAIST

Data security in sensitive remote services



Send confidential data to a remote service

Messaging

Navigation

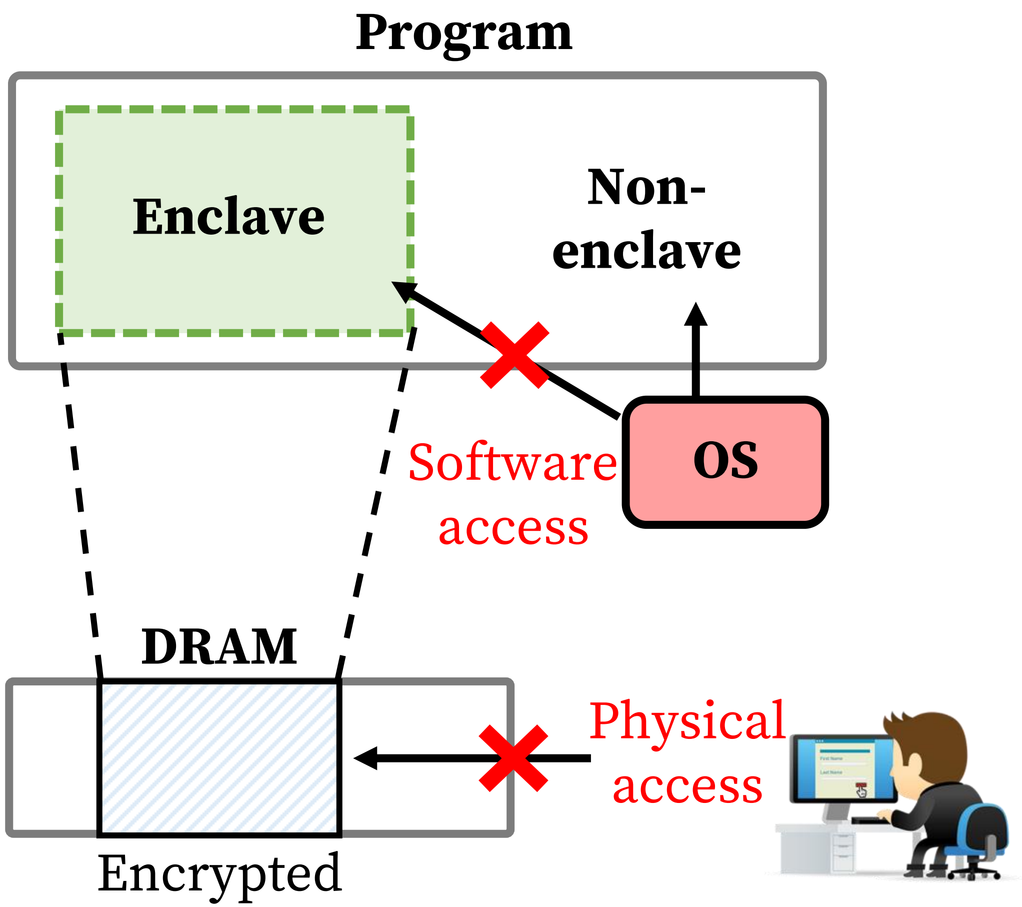
DNA testing

Machine learning

Intrusion detection

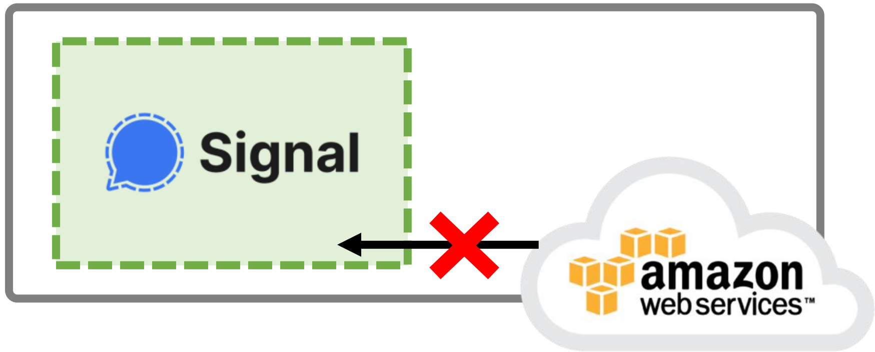
Guarantee that data cannot be collected!

SGX is designed to secure remote data



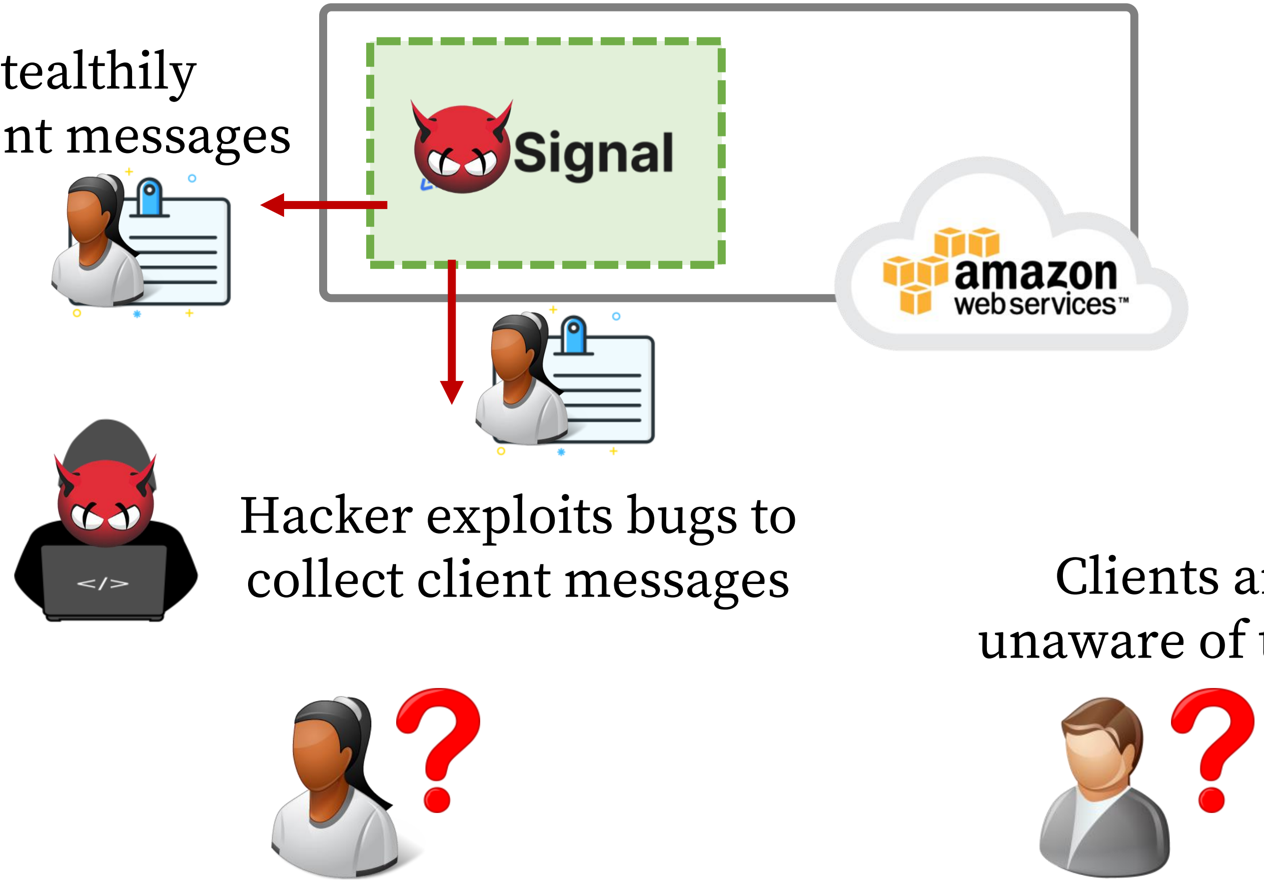
SGX secures remote data from clouds

Signal uses SGX;
Amazon cannot access
Signal's service



Client messages
through Signal are
safe from Amazon

SGX does not secure data from untrusted code

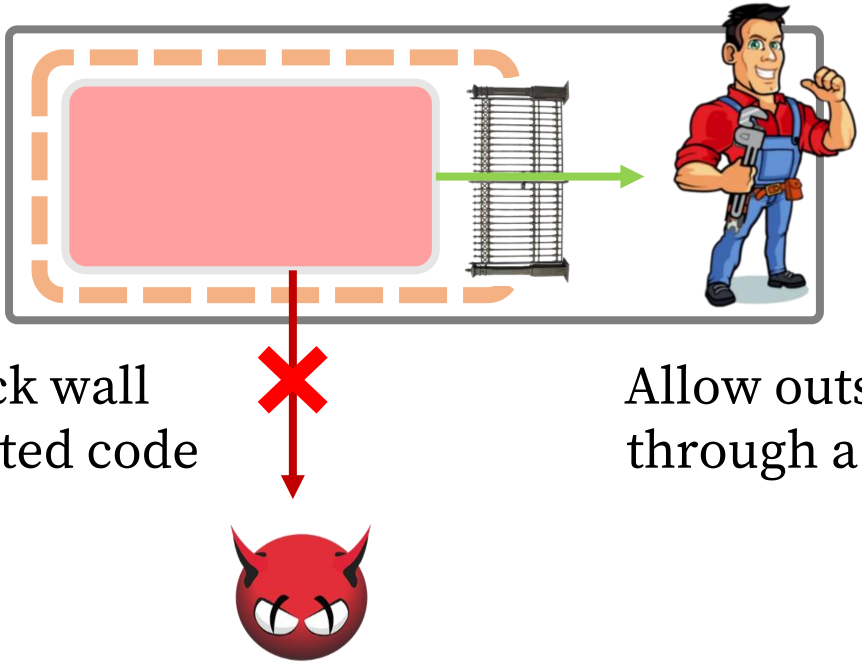


Signal stealthily collects client messages

Hacker exploits bugs to collect client messages

Clients are unaware of theft!

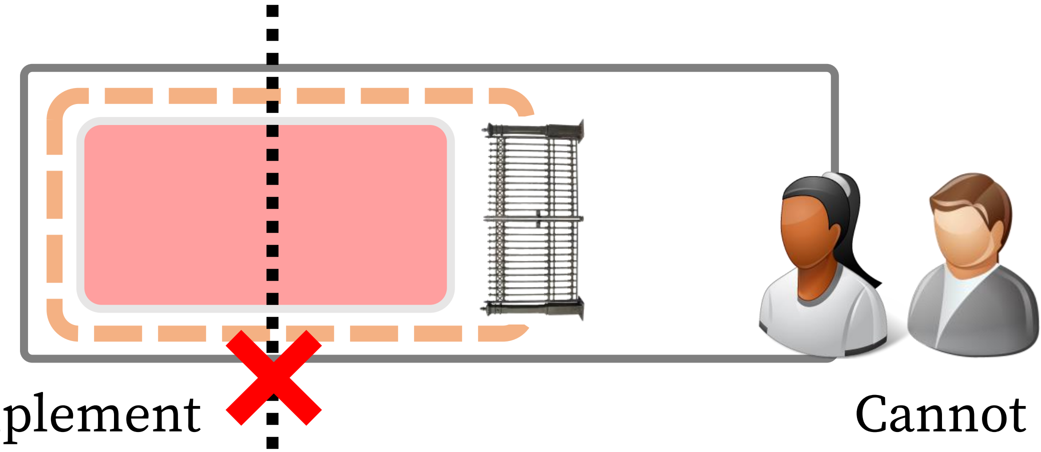
Software fault isolation restricts untrusted code



Create a brick wall around untrusted code

Allow outside access only through a controlled gate

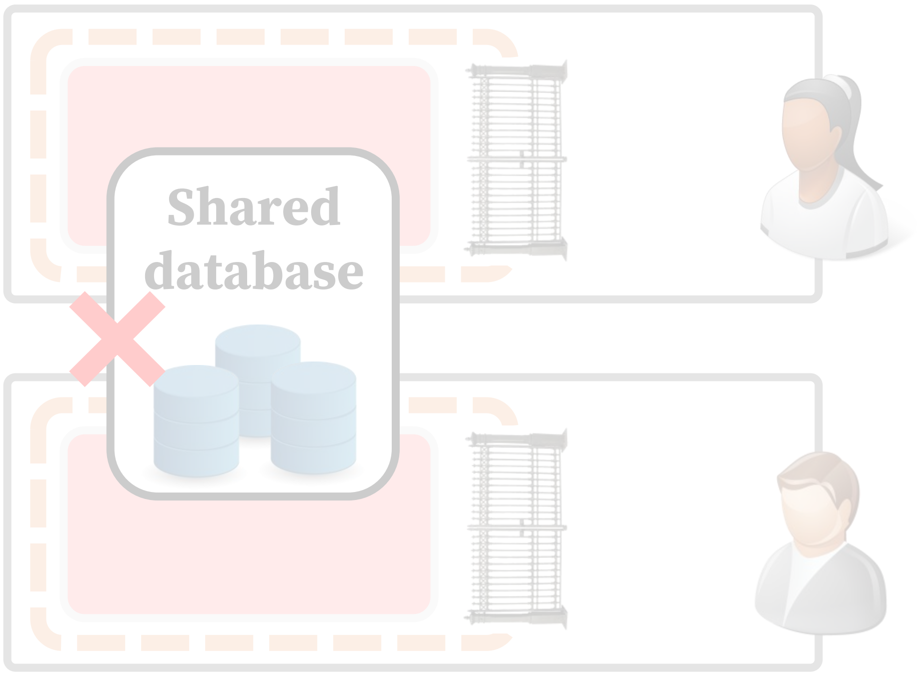
Native Client SFI requires multiple processes



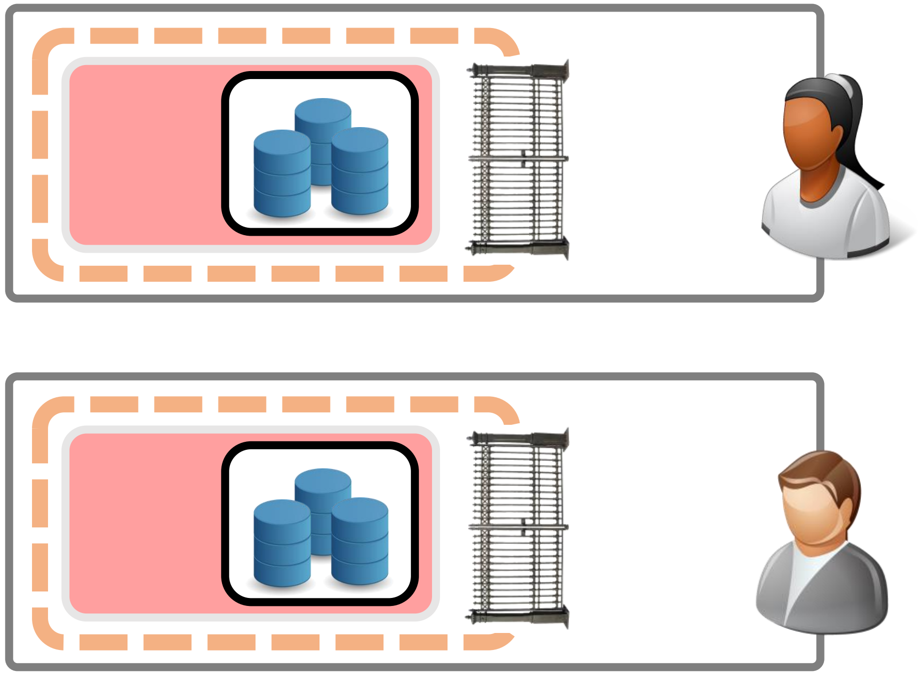
Does not implement in-process isolation

Cannot serve multiple clients in a process

Multiple processes consume a lot of memory



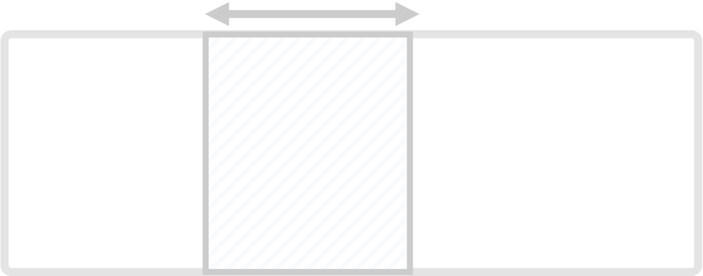
Lack efficient and secure
inter-process memory sharing



Must replicate common
data in each process

High memory use reduces enclave performance

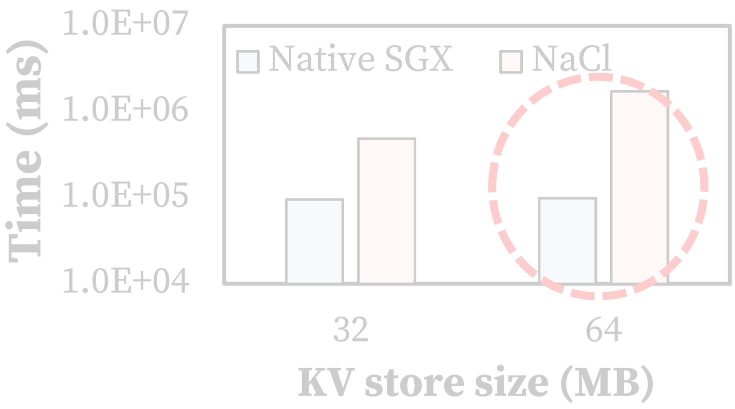
SGX memory is only 256MB

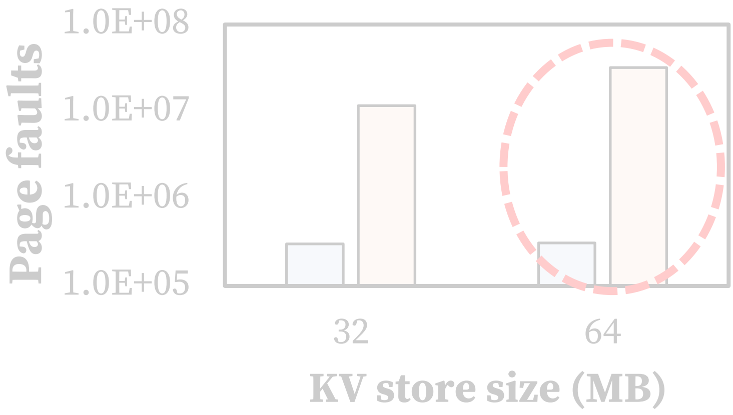


Memory usage over 256 MB
incurs expensive page faults

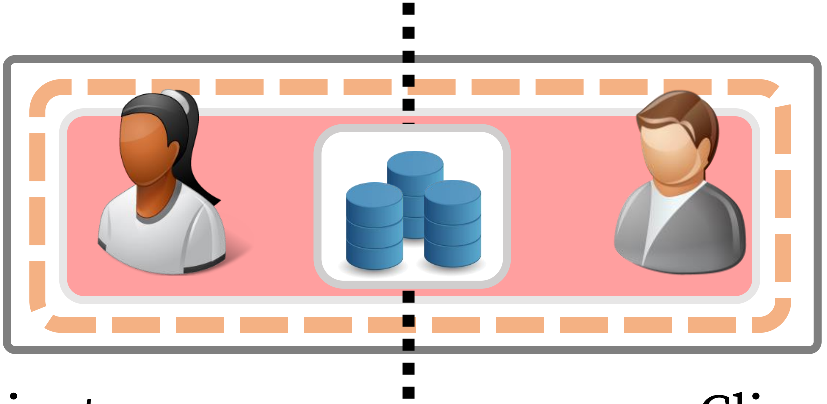
Native Client (NaCl) SFI can be 16 times slower than native SGX!

Key-value store with 8 clients





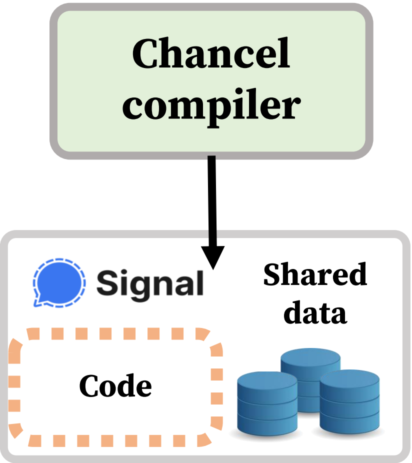
Chancel implements efficient multi-client SFI



Multiple clients are served by a process

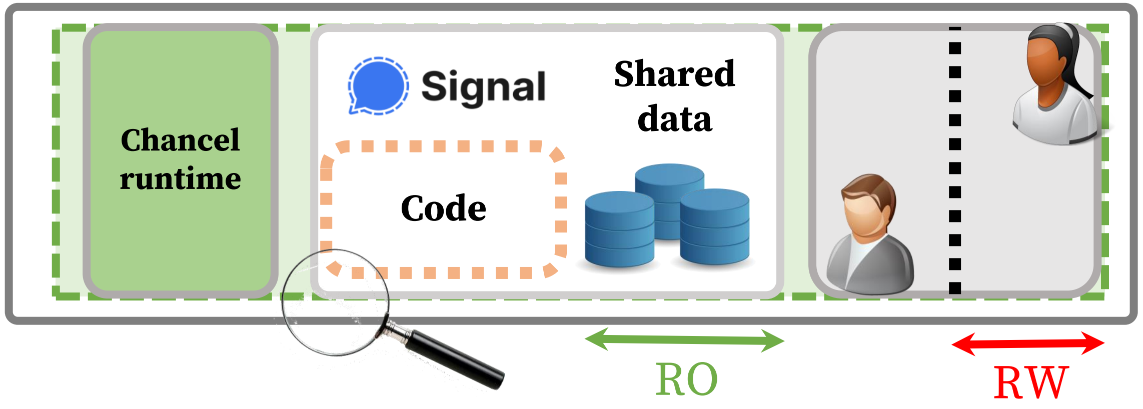
Clients securely access shared memory

Chancel's design



1. Automated program instrumentation

Offline stage



2. Enclave creation and program loading

3. Secure client bootstrapping

4. Multi-client SFI enforcement

Online stages

1. Automated program instrumentation

Registers = {RAX, … , R12, R13}

Compiler reserves
registers R14 and R15

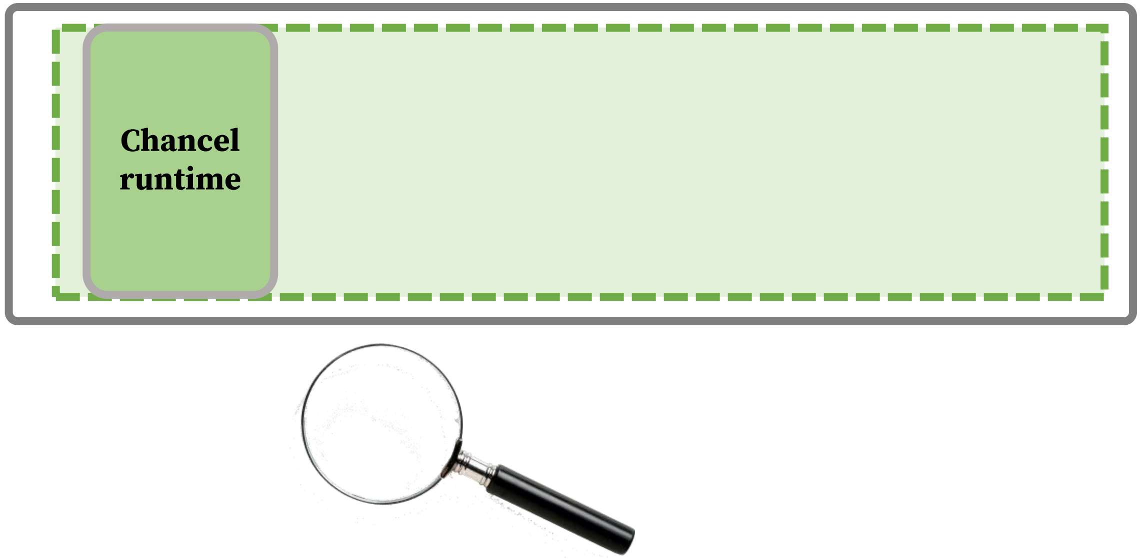
Before:
write at X

After:
if X < R14 + thread size,
write at X

Compiler bounds writes relative to R14
and reads relative to R14 or R15

2. Enclave creation and program loading

Create enclave installed with Chancel's trusted runtime

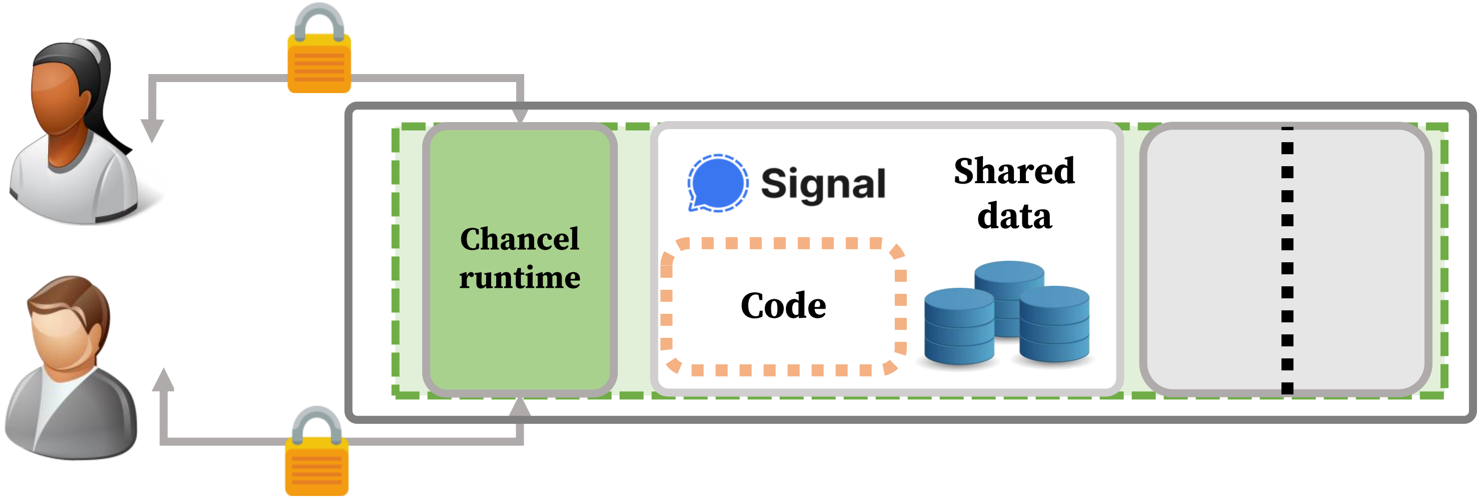


Thanks to validation, Chancel even supports proprietary code!

Validate instrumentation using a binary disassembler

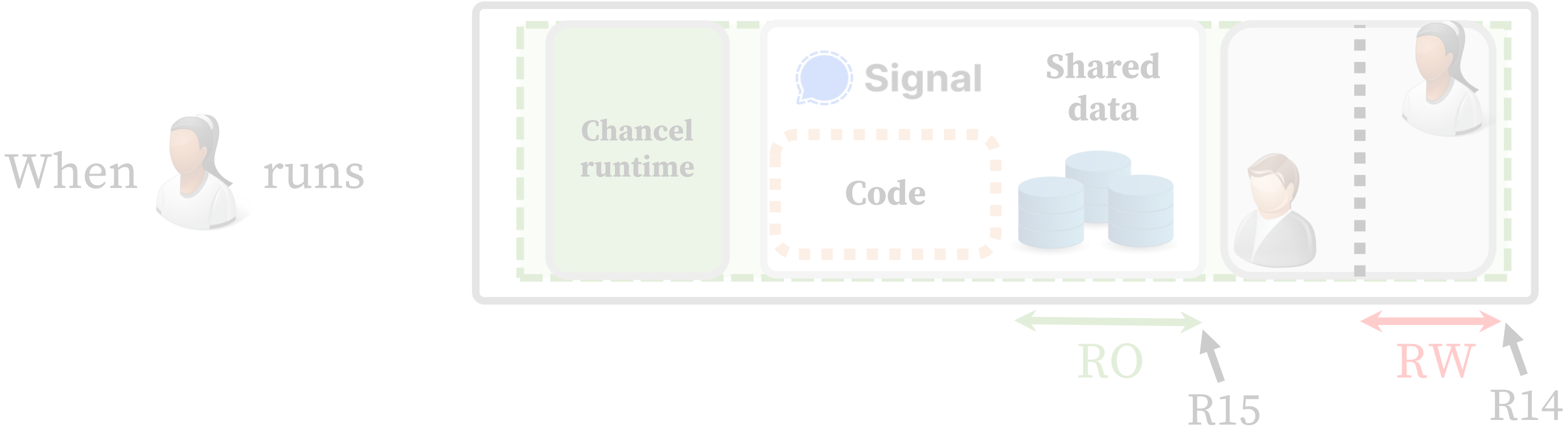
3. Secure client bootstrapping

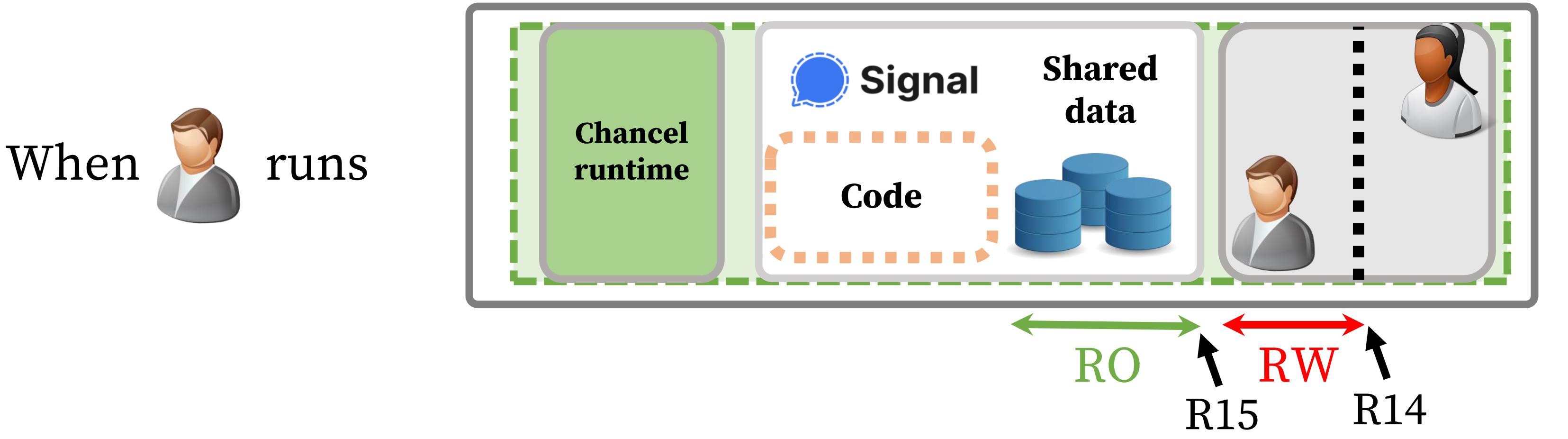
Clients attest Chancel and transmit their data through encrypted channels



Store each client's data in a different enclave thread

4. Multi-client SFI enforcement

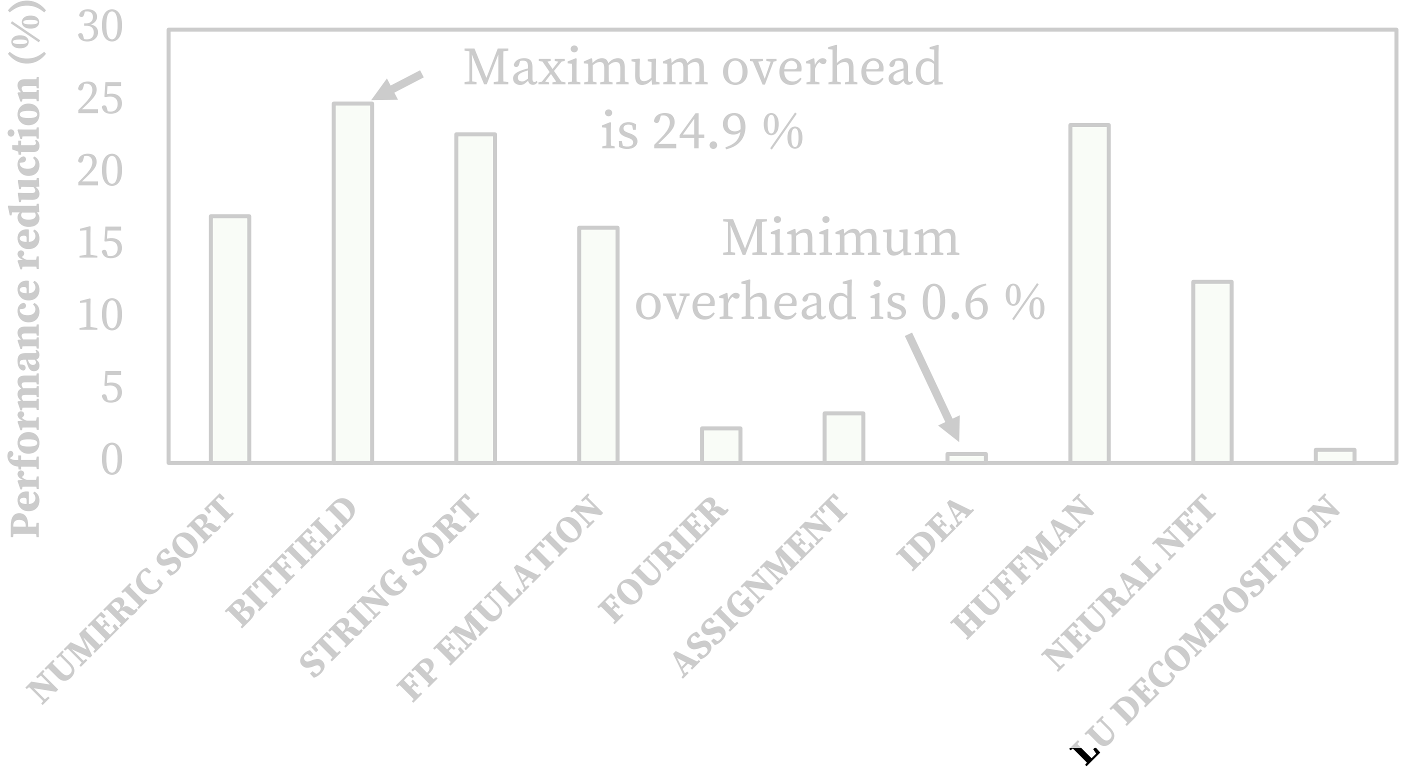




Overhead over native SGX

Ran all applications in Nbench, a popular SGX CPU and memory benchmark

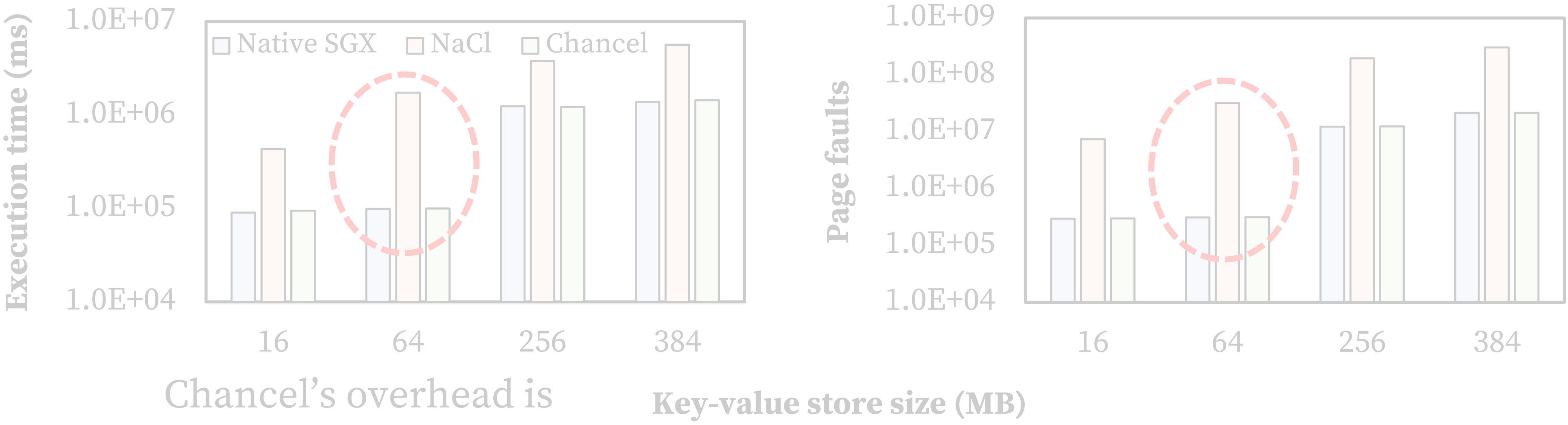
Despite high security guarantees, Chancel's overhead is modest!



Benefit over Native Client

100,000 "GET" requests to ShieldStore key-value store from 8 clients

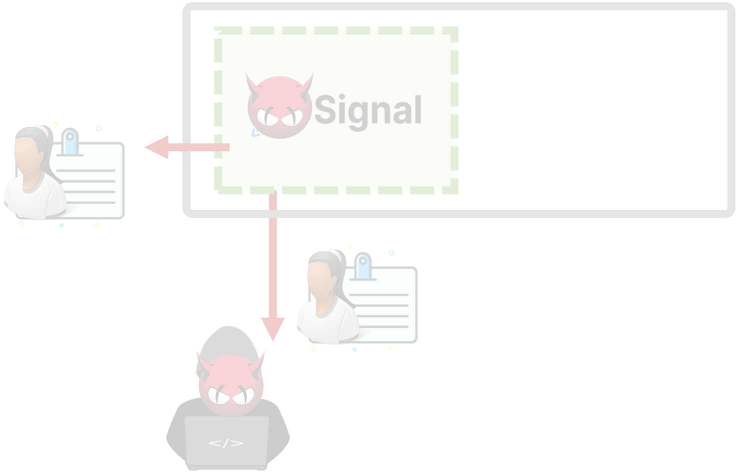
Across diverse applications, Chancel outperforms multi-process Native Client (NaCl) by up to 21 times!



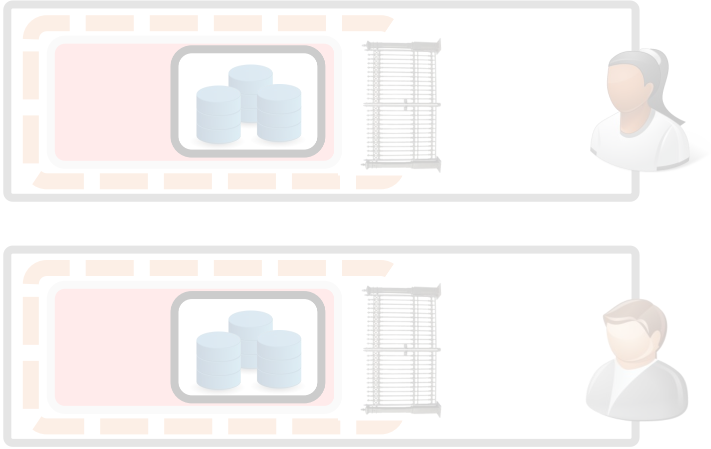
Chancel's overhead is 1.1 – 8.4% over native SGX

Summary and conclusion

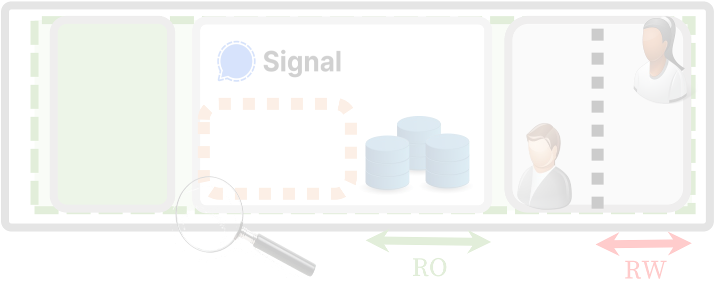
SGX does not secure remote data from untrusted code



Multi-process SFI is slow in multi-client enclaves



Chancel’s SFI is up to 21 times faster than multi-process SFI



Thank you!